Vision-Guided Autonomous Stair Climbing

Yalin Xiong Larry Matthies
Machine Vision Group
Jet Propulsion Laboratory
Pasadena, CA 91109

Abstract

The Tactical Mobile Robot (TMR) program calls for autonomous mobility in an urban environment. Among all man-made structures which pose as barriers for a mobile robot, stairs are the most obvious and ubiquitous structure an autonomous urban robot needs to be able to handle.

The urban II chassis by IS Robotics provides a mechanically simple and elegant way to enable the stair-climbing capability. This paper addresses the issue of using computer vision techniques to control the vehicle automatically when climbing stairs. The algorithm is based on detecting stair edges from a monocular sequence. It is shown that the position and orientation of the stair edges can be used to compute the 3D orientation and position of the vehicle relative to the stairs. Robust estimation of those parameters is the key to reliable performance. A combination of techniques such as median filtering, histogram peak finding, outlier rejection and weighted average are shown to be effective in practice. At the end, we briefly explain our work-in-progress on automatic corner-turning in climbing multiple flights of stairs.

Figure 1: Urban II Vehicle

Figure 2: Mounting and Climbing Stairs

1 Introduction

The urban environment poses a great challenge to autonomous mobile robots. From the mobility point of view, many man-made structures such as curb or stair are difficult for small robot to overcome. The main objective of the Tactical Mobile Robot (TMR) program is to enable small packable robots to move effectively in an urban environment. One of the first new issues is to climb stairs automatically.

The urban II chassis by IS Robotics ([4]) provides a simple mechanism to climb stairs and curbs. Figure 1 shows the overall mechanical design of the vehicle. The vehicle has two main tracks along its body. Those two tracks are used for driving and steering. In the front of the vehicle, there are two articulated arms with tracks as well. The vehicle uses the two arms to mount a stair. When the vehicle is on the stairs, the arms are straightened out to provide more stable support. Figure 2 illustrates the mechanism in which the vehicle mounts and climbs the stairs.

In order for the vehicle to climb the stairs automatically, two critical parameters must be estimated robustly and continuously during the climbing: the angle between the stair orientation and vehicle heading direction, and the position of the vehicle on the stairs with respect to the center of stairs, i.e., whether the vehicle is too close to the left or right boundary of the stairs. For the convenience of illustration, we refer to the first parameter as the "offset angle", and the

Figure 3: Stair Edges

second parameter as the "offset position".

We demonstrate that both parameters can be estimated from stair edges detected in a 2D image. Stair edges are defined as those 3D lines parallel to steps. Figure 3 shows the edges of a flight of stairs in grey lines. In 3D, they are parallel with each other, though they do not belong to one single plane because of the zigzag profile of typical stairs. In the next section, we will illustrate that, if we assume that the vehicle body is approximately parallel to the 3D stair edges, we can compute the offset angle and the offset position in a closed-form solution without any 3D information.

Thus it is critical to detect those stair edges robustly. Unfortunately stairs come in different styles, different contrasts, and different materials. To make the situation worse, outdoor stairs frequently have shadows cast onto them. All of these conditions complicate the detection of stair edges significantly. It is our belief that the stair edge detection has to be specialized to robustly extract near-parallel edges. In our implementation, we take a minimum commitment approach to start with a paranoid edge detector, which detect a large number of potential candidates. We then filter those candidate edges through histogram peak finding, median filtering, outlier rejection, and aggressive edge linking.

To extend the current capability to climbing multiple flights of stairs, the vehicle needs to go around the corner automatically and mount the next flight. We will briefly describe three strategies from simple scripting to complicated target recognition/tracking. The work on this is in progress. The experimental results we show are done by scripting.

2 Stair-Climbing Algorithm

Stairs such as in Figure 4 can be very different in appearance. The objective is to extract from the monocular images the offset angle and the offset position parameters *robustly*. We present in this section an algorithm based on detecting stair edges and extracting offset parameters from the stair edges. The stair-climbing algorithm is broken down into three steps:

Figure 4: Appearance of Stairs in Real World

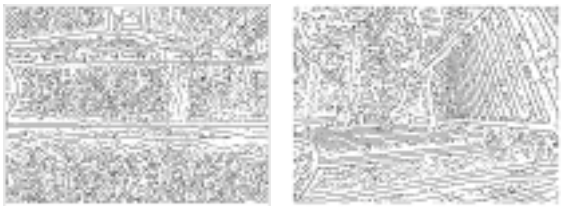

Figure 5: Thinning of Edge Pixels

1. Edge detection and linking to extract stair edges. This subsystem must deal with all the variance of appearance of stairs, and be able to extract valid stair edges under most lighting conditions.

2. Recovery of offset parameters: From a set of stair edges, compute the offset angle and offset position. This subsystem must be robust against outliers since the edge detection and linking will not be perfect.

3. Steering of the vehicle: Steering the vehicle according to the offset parameters.

The following subsections explain in detail each of the steps.

2.1 Edge Detection and Linking

Generic edge detector ([2, 1]) will not work very well on stair images such as Figure 4. When the stair is under shadow or stair edges are rough, we only get broken edges from which it is hard to extract any useful information. Instead, we take a least-commitment approach in the initial edgel detection by ommiting the threholding entirely. This will preserve potential stair edges under shadow or when the stair has low contrast. Figure 5 shows the thinned edge pixels.

Such a approach will potentially generate hundreds of edges or more, therefore increasing the difficulties

in filtering out spurious ones. Our argument is that filtering out spurious ones is always better than prematurely rejecting real stair edges. In addition, comparing to the generic edge detectors, we do have some more constraints that improve filtering:

- Stair edges are straight lines.
- Stair edges are close to parallel to each other and close to be horizontal.
- Multiple stair edges nearby can be merged into a single one.
- Stair edges are long.

The edge detection and linking algorithm filter out spurious edges by sequentially applying the above four constraints:

1. Straight line constraint: When linking neighboring edge pixels into an edge, we run a line-parameter estimator to examine whether the new pixel is collinear with the existing edges. This constraint guarantee that all the edges are straight lines.
2. Dominant orientation constraint: For all straight edges found so far, compile a histogram of number of edge pixels against their orientations. Choose the peak orientation within [-45, 45] degrees from the horizontal orientation. Eliminate all other edges that are too far from the peak orientation. Note that parallel 3D lines are usually not mapped into parallel 2D lines, but in general, the orientations will not vary much.
3. Linking multiple edges: It is inevitable that some stair edges will be broken due to shadow or other reasons. This step links those collinear small edges into one single long edge.
4. Length of stair edges: After all above filtering, the short edges are eliminated. The threshold used was 1/4 of image width.

The first and third step in the above procedure are worth more elaborations. In enforcing the straight line constraint during edge tracking, we repeatedly estimate the parameters of a fitted line every time a new pixel is added, and when distance between the next edge pixel and the fitted line exceeds a limit, the edge tracking stops in that direction. Figure 6 shows the results after tracking straight lines and applying the dominant orientation constraint. Though the results from the edge tracking are sensitive to starting edgels, the subsequent aggressive edge linking makes them less sensitive.

Figure 6: Tracking Straight Lines

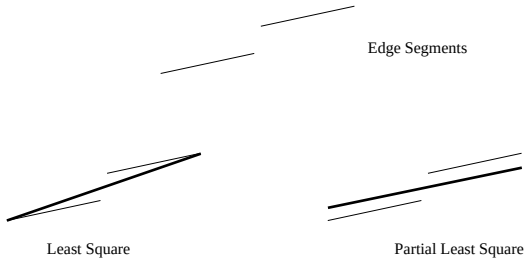

Figure 7: Partial Least Square for Edge Linking

When linking two edges that have the same orientation but are slightly shifted vertically, the least square fitting does not make any sense because the slight vertical shift will cause the orientation of the merged line to be off. Instead, we perform a partial least square fitting by adjusting only the position parameter while keep orientation parameter the same. Figure 7 shows the difference between a general least square fitting and our partial least square fitting. This situation occurs frequently when there are several parallel lines in close distance on the stair steps, and only a portion of each of the lines is detected.

Mathematically, let us assume the ith straight edge has the following statistical parameters: N^i, S^i_x, S^i_y, S^i_{xx}, S^i_{xy} and S^i_{yy}, where N^i is the number of pixels of the edge, S^i_x is the sum of all x coordinate of the pixels, so on as in [3]. If we represent the fitted line by the equation $y = k^i x + b^i$, then by the least square method:

$$k^i = \frac{N^i S^i_{xy} - S^i_x S^i_y}{N^i S^i_{xx} - (S^i_x)^2}, \quad (1)$$

$$b^i = \frac{S^i_{xx} S^i_y - S^i_x S^i_{xy}}{N^i S^i_{xx} - (S^i_x)^2}, \quad (2)$$

and the average square residue is

$$r^i(k^i, b^i) = (S^i_{yy} + (k^i)^2 S^i_{xx} + N^i (b^i)^2,$$

$$-2k^i S^i_{xy} - 2b^i S^i_y + 2b^i k^i S^i_x)/N^i. \quad (3)$$

When merging two edges with almost identical slope, for example edge ith and jth, we estimate the slope by a simple weighted average:

$$k^{ij} = (k^i N^i + k^j N^j)/(N^i + N^j), \quad (4)$$

and the position parameter is fitted by least square:

$$b^{ij} = (S^i_y + S^j_y - k^{ij}(S^i_x + S^j_x))/(N^i + N^j), \quad (5)$$

and the average square residue after the merging is:

$$r^{ij} = (N^i r^i(k^{ij}, b^{ij}) + N^j r^j(k^{ij}, b^{ij}))/(N^i + N^j). \quad (6)$$

The edge linking algorithm has the following steps:

1. Sort the list of straight lines by their length in descending order. Denote the list as L.
2. For each line i, find the set of lines L_i whose member are parallel to line i.
3. Sort the list L_i according to the distance between the member and line i in ascending order. The distance is in the direction normal to the line orientation. Eliminate those elements with large distance from list L_i.
4. Sort the list L_i according to the gap between the member and line i in ascending order. The gap is in the direction of the line orientation.
5. Try to merge line i with each member j of the list L_i:
   (a) If the gap between line i and j is too large, continue.
   (b) Compute the k^{ij}, b^{ij} and r^{ij}.
   (c) If the residue r^{ij} larger than a threshold, continue;
   (d) Replace line i by the merged line and delete line j from list L.
6. If the length of the merged line i is less than a threshold, reject it. Otherwise, add the line to a list of stair edges E. goto step 2.

Figure 8 shows the stair edges after all the filtering and linking.

Figure 8: Merged Straight Edges

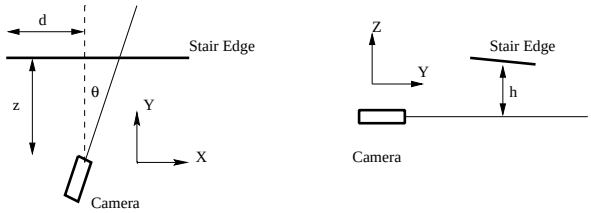

Figure 9: Mapping 3D Stair Edges onto Images

2.2 Robust Recovery of Offset Parameters

In order to describe the extraction of offset parameters, let us take a look at how a 3D stair edge is projected onto the image. Suppose there is a 3D stair edge, and a camera is looking at the edge at an offset angle of θ. The distance between the camera and the edge is z, the height of the stair edge is h, and point location in the edge is represented by d as in Figure 9. Without loss of generality, we also assume the focal length of the camera is 1. Then the projective mapping of any 3D point on the line onto the camera image plane can be represented by a homogeneous transform:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1 & x_0 & 0 \\ 0 & y_0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d \\ z \\ h \end{bmatrix} \quad (7)$$

where $s\theta = \sin(\theta)$, $c\theta = \cos(\theta)$ and x_0 and y_0 are the projection center of the image plane.

In the 2D image plane, the mapped location of any point on the edge is

$$x = \frac{X}{Z} = x_0 + \frac{dc\theta - zs\theta}{ds\theta + zc\theta}, \quad (8)$$

$$y = \frac{Y}{Z} = y_0 - \frac{h}{ds\theta + zc\theta}. \quad (9)$$

By eliminating variable d from the above two equations, we then obtain the equation for a 2D straight line which is the image of the 3D stair edge:

$$y - y0 = \frac{hs\theta}{z}(x - x_0) - \frac{hc\theta}{z}. \quad (10)$$

This equation directly relates the parameters of 2D and 3D stair edges.

If we have already detected the stair edge in the image, we can intersect the edge with the image center column $x = x_0$, we then have

$$y_m - y_0 = -\frac{hc\theta}{z}, \tag{11}$$

where y_m is the Y-coordinate of the intersection point, and the slope of the 2D stair edge measured from the image should satisfy the following constraint:

$$k = \frac{hs\theta}{z}. \tag{12}$$

From the above two equations we have

$$\theta = \text{atan}(-k/(y_m - y_0)), \tag{13}$$

which elegantly states the relationship among the offset angle and the line position and orientation.

From Eq. 8, the distance d as a function of the x-coordinate in image can be represented as

$$d(x) = z\frac{(x - x_0) + \tan(\theta)}{1 - (x - x_0)\tan(\theta)}. \tag{14}$$

Without knowing the distance z (scale parameter), it is impossible to estimate the horizontal distance $d(x)$. But we can estimate the *ratio* between the distance to the left and right endpoints of the stair edge using the above equation, i.e.

$$q = d(x_l)/d(x_r), \tag{15}$$

where x_l and x_r are the x-coordinate of the left and right endpoints respectively. This ratio q will be used to steer the vehicle to the center of the stairs.

Considering the special case when $h = 0$, Eq. 10 states that the 2D edges are always at the same orientation (horizontal) regardless of the offset angle. Another way to look at this singularity issue is that, from Eq. 13, it is obvious that when y_m is near to the image center y_0, the estimation of the offset angle is numerically unstable. In practice, for any candidate stair edge, if its y_m is near the the image center, it is not used for estimating the offset angle though it can be used to estimate the ratio q once the offset angle is computed from other stair edges.

Though Eq. 13 and 15 establish a simple way to estimate the offset parameters, we still need address the issue of robustness given that there will be outliers in the stair edges. We rely mostly on median filtering to reject outliers:

Figure 10: The Final Set of Stair Edges

1. Compute q and θ from every stair edge.
2. Reject those edges whose q values are incorrect, e.g. the left and right endpoints are on the same side of the vehicle.
3. Compute the median of θ by sorting the valid θ values.
4. Reject those edges whose θ values are far from the median.
5. Weight average θ values. The weights are proportional to $1/(y_m - y_0)$.
6. Recompute q for every edge using the averaged θ. Choose the median of all q values to be the offset position ratio.

Figure 10 shows the final set of stair edges after the above procedure.

The main assumption under which Eq. 13 and 15 hold valid is that the plane defined by the nodal point and the center scanline in the image is parallel to all the stair edges. When the vehicle is on the stairs, the vehicle body plane is approximately parallel to all the stair edges. Therefore, we can "rectify" the raw image so that the assumption is satisfied. In case where the vehicle body rolls with respect to the stair edges, those equations hold approximately true as long as the roll angle is not significant. If vehicle roll angle can be independently estimated, the roll angle can also be compensated, though in practice it is rarely an issue.

2.3 Steering of Vehicle on Stairs

The steering of the vehicle is determined by two criterions:

- Alignment: The vehicle needs to be aligned with the stairs in orientation. The objective is to servo the steering so that the offset angle θ is small in magnitude.

- Centering: The vehicle needs to stay close to be center of the stairs. The objective is to servo the steering so that the offset position ratio q is close to 1.

For both criterions, we use a simple linear mapping from the deviations to the steering angles. The final steering angle is a weighted average of the two.

The alignment criterion has higher priority than the centering criterion. Therefore, when the offset angle has large magnitude, the vehicle is steered purely by the first criterion in order to avoid catastrophic failure. The centering criterion is to provide a better safety buffer, and is taken into consideration only after the vehicle is basically aligned with the stair.

Another reason the centering criterion is given less consideration is that the accuracy of q is limited by factors such as the field of view of the camera (edges clipped), relatively inaccurate endpoint position of the detected edges, and so on. In practice, we also enforce a temporal consistency checking on q values.

3 Corner-Turning for Multiple Flights of Stairs

In order for the vehicle to climb multiple flights of stairs, it needs to navigate around the landing corner, and align with the next flight of stairs. This is more of a general navigation issue, whose solution ranges from simple scripting to elaborate target recognition and tracking. Here are three approaches with increasing sophistication and difficulties:

- Scripted Navigation:If the physical dimension of the landing corner is known, once the vehicle lands, it can go through a scripted sequence of motions to get around the corner. It is the simplest approach but requires *apriori* knowledge of the particular stairwell.
- Wall Following: After the vehicle lands, it can follow the boundary (wall) of the landing corner using 3D sensors such as stereo or laser rangefinder. It is a more general approach than the scripted navigation, but it can not handle stairwells with extra exits.
- Target Recognition/Tracking: After the vehicle lands, it can pan the sensor around to locate where the next flight of stair is, plan the path and avoid obstacle when navigating toward it. This is the most general and difficult approach.

Figure 11: Climbing Two Flights of Stairs

Figure 12: Climbing Stairs Under Shadow

4 Experiments

We conducted extensive testing of stair climbing under various lighting and shadow conditions on various types of stairs. The overall system robustness is satisfactory. The following two sequences of images were shot when the vehicle was climbing a multiple flights of narrow and steep fire-escape stairs with the self-casting shadow.

Acknowledgments

The authors would like to thank the entire hardworking team of BAA 97-20 to make this work possible.

References

[1] Dana Ballard and Christopher Brown. *Computer Vision*. Prentice-Hall Inc., 1982.

[2] J. Canny. A computational approach to edge detection. *IEEE Trans. PAMI*, 8(6):679–697, November 1986.

[3] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[4] IS Robotics. http://www.isr.com/research/urban.html. 1999.